Building Decoupled Architectures

TIGHT VS LOOSE COUPLING · AMAZON SQS · AMAZON SNS · AMAZON MQ · ASYNCHRONOUS MESSAGING STUDY & REVIEW

01 MUST KNOW

if you remember five things, remember these

01 **Tight vs. loose coupling: tightly coupled** components depend directly on one another — a single failure or change forces failures or changes in the others, and scaling means editing code to add connections. **Loose coupling** inserts an intermediary between components so they scale and fail independently; it can be **synchronous** (ELB, microservices) or **asynchronous** (Amazon SQS, SNS, MQ).

02 **Amazon SQS = queue (point-to-point):** a **producer** puts messages in a **queue**; a **consumer** *polls* the queue (pull), processes each message, then **explicitly deletes** it. Fully managed; messages **persist** until deleted or the retention period expires. One message is consumed by one consumer — one-to-one.

03 **Amazon SNS = topic (pub/sub):** a **publisher** sends one message to a **topic**; SNS **pushes** it to all **subscribers** — one-to-many. SNS does **not persist** messages (deleted after publishing). Subscribers include email, SMS, mobile push, HTTP/HTTPS, Lambda, SQS queues, and Kinesis Data Firehose.

04 **The SQS knobs the exam loves: standard** queue = at-least-once, best-effort order, near-unlimited throughput; **FIFO** = exact order, exactly-once, up to 300 (or 3,000 batched) API calls/s. **Visibility timeout** hides an in-flight message (default 30 s, max 12 h). **Long polling** cuts empty responses and cost. Unprocessable messages go to a **dead-letter queue (DLQ)**.

05 **Amazon MQ = managed broker for hybrid:** a fully managed **Apache ActiveMQ / RabbitMQ** broker supporting queues *and* topics plus open protocols (JMS, AMQP, MQTT, STOMP, NMS, WebSocket). Use it to connect **on-premises apps with the AWS Cloud** or migrate existing brokers **without rewriting messaging code** — not the pick for brand-new cloud-native apps.

02 KEY TERMS

TERM	DEFINITION
Decoupling	Separating the components of a system so they can operate independently, increasing resiliency and agility
Tight coupling	Components are so dependent that a change or failure in one forces changes or failures in others; scaling requires code updates
Loose coupling	Reduces dependencies by inserting an intermediary between components so they scale and fail independently; synchronous or asynchronous
Producer / consumer	Point-to-point roles: the producer puts messages in a queue; the consumer pulls and processes them (Amazon SQS)
Publisher / subscriber	Pub/sub roles: the publisher sends messages to a topic; subscribers receive every message pushed from that topic (Amazon SNS)
Message queue	A temporary repository that holds messages until a consumer retrieves and deletes them
Topic	A named access point a publisher posts to and subscribers register with; messages are pushed to all subscribers
Visibility timeout	Period during which a received SQS message is hidden from other consumers (default 30 s, maximum 12 h)
Dead-letter queue (DLQ)	A queue associated with a source queue that receives messages that could not be processed successfully
Long polling	Receive-message wait time greater than 0 (max 20 s); queries all servers and waits, reducing empty responses and cost

TERM	DEFINITION
Fanout	A message sent to an SNS topic is replicated and pushed to multiple SQS queues, HTTP endpoints, or email addresses for parallel processing
Amazon MQ	Managed message broker for Apache ActiveMQ and RabbitMQ; loosely couples on-premises and cloud apps using open messaging standards

03 TIGHT COUPLING VS. LOOSE COUPLING

the central trade-off of the module

TIGHT COUPLING

components directly dependent — fragile

- Communication is synchronous — a component sends a request and **waits** for a reply before proceeding
- One component's failure takes down the system; a fix or update means whole-system downtime
- Scaling out means updating code to add connections and route traffic to new instances
- In a monolith, one slow function can degrade or stop the entire application

LOOSE COUPLING

an intermediary absorbs change and failure

- An intermediary (ELB, queue, topic, or broker) sits between dependent components
- The intermediary handles failover and scaling automatically; components scale and fail independently
- Adding a server needs far fewer connections (e.g., only two — to ALB1 and ALB2)
- Synchronous options: ELB, microservices · Asynchronous: Amazon SQS, SNS, MQ

04 AMAZON SNS VS. AMAZON SQS

push/pull · one-to-many/one-to-one · persistence

DIMENSION	AMAZON SNS	AMAZON SQS
Messaging model	Publisher–subscriber	Producer–consumer
Distribution model	One to many	One to one
Delivery mechanism	Push (passive)	Pull (active — consumer polls)
Message persistence	No — deleted after publishing	Yes — until deleted or retention expires

05 CHOOSING SQS · SNS · MQ

match the service to the workload

FACTOR	AMAZON SQS	AMAZON SNS	AMAZON MQ
Applicability	Cloud-centered apps	Cloud-native apps	Hybrid apps; broker migration
Messaging model	Producer–consumer	Publisher–subscriber	Both models
Programming API	Amazon SQS API	Amazon SNS API	Industry-standard broker APIs
Pricing model	Pay per request	Pay per request	Pay per hour + pay per GB

06 FANOUT: ONE SNS TOPIC → MANY SQS QUEUES

parallel image resizing

01 Mobile client uploads an image to an **S3 bucket** → **02** S3 **Event Notifications** publish the object URL to an **SNS topic** → **03** The topic **fans out** the message to three SQS queues (thumbnail, mobile, web) → **04** Each app polls its own queue and resizes independently in its Auto Scaling group → **05** Each app stores its result in a converted-images **S3 bucket**

07 EXAM TRAPS

where the knowledge check gets you

- × "Amazon SNS stores messages until a subscriber reads them." — No. SNS **does not persist** messages; it deletes each one after publishing to subscribers. It is **Amazon SQS** that persists messages until the consumer deletes them or retention expires.
- × "Short polling is the better, cheaper default." — Backwards. Short polling queries only a subset of servers and returns immediately (often empty), raising cost. **Long polling** queries all servers and waits, reducing empty responses — preferable in almost all cases.
- × "SQS pushes messages to its consumers." — SQS is **pull**: consumers *poll* the queue. **Push** delivery is the signature of Amazon SNS.
- × "Receiving an SQS message removes it from the queue." — No. The message stays in the queue; the consumer must **explicitly delete** it. If it is not deleted before the visibility timeout expires, the message becomes visible again and may be processed twice.
- × "Standard queues guarantee order and exactly-once delivery." — No. **Standard** = at-least-once (possible duplicates) and best-effort ordering. Only a **FIFO** queue guarantees exact order and exactly-once processing.
- × "Use Amazon MQ for new cloud-native applications." — For new cloud apps AWS recommends **Amazon SQS and SNS**. Amazon MQ is for **hybrid** integration and **migrating existing** ActiveMQ/RabbitMQ brokers without rewriting code.
- × "Loose coupling means asynchronous messaging." — Loose coupling can be **synchronous** (ELB at the infrastructure level, microservices at the application level) **or** asynchronous (SQS, SNS, MQ).
- × "A standard queue can use a FIFO dead-letter queue." — No. The DLQ of a **standard** queue must be standard, and the DLQ of a **FIFO** queue must be FIFO.

08 SELF-QUIZ

answers are upside-down below

Q1 The producer and consumer roles belong to which messaging model — point-to-point or publish/subscribe?

Q2 Which service uses publishers, subscribers, and topics?

Q3 Does Amazon SQS deliver messages by pushing them or by having consumers pull them?

Q4 Does Amazon SNS persist messages after publishing them?

Q5 What is the maximum size of a standard Amazon SQS message?

Q6 Which SQS queue type preserves exact ordering and processes each message exactly once?

Q7 After a consumer successfully processes an SQS message, what must it do so the message is not processed again?

Q8 What is a dead-letter queue used for?

Q9 Which service does AWS recommend to integrate on-premises apps with cloud apps or to migrate an existing message broker?

Q10 Sample exam: to keep the number of empty responses from SQS polling requests to a minimum, what should a solutions architect configure?

ANSWER KEY (rotate the page)

Q1 point-to-point (used by Amazon SQS) **Q2** Amazon SNS (publish/subscribe) **Q3** pull — consumers poll the queue **Q4** no — SNS deletes each message after publishing it to subscribers **Q5** 256 KB (up to 2 GB using the SQS Extended Client Library for Java) **Q6** a FIFO queue **Q7** explicitly delete the message from the queue **Q8** to store messages that cannot be processed successfully **Q9** Amazon MQ (Apache ActiveMQ or RabbitMQ) **Q10** increase the receive-message wait time (enable long polling)